Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN

José García-Fanjul, Javier Tuya, Claudio de la Riva [1,2]

Computer Science Department
University of Oviedo
Gijón, Spain

Abstract

Generating test cases for compositions of web services is complex, due to their distributed nature and asynchronous behaviour. In this paper, a formal verification tool – the SPIN model checker – is used to generate test suite specifications for compositions specified in BPEL. A transition coverage criterion is employed to define a systematic procedure to select the test cases. The approach is applied to the "loan approval" sample composition.

Key words: web service compositions, model-based testing, structural testing, model checking.

[1] This work is supported by the Ministry of Science and Education (Spain) under the National Program for Research, Development and Innovation, projects IN2TEST (TIN2004-06689-C03-02) and REPRIS (TIN2005-24792-E)
[2] Emails: {jgfanjul, tuya, claudio}@uniovi.es

1 Introduction

From the inception of the Web, a growing number of companies have tried to use it as a new commercial channel. To start with, the most visible approach to leveraging this technology was the deployment of publicly available web-based shops, committed to collecting orders made by consumers. At an early stage, businesses also saw the opportunity to establish commercial electronic relations among each other. To do so, they needed to agree on a certain protocol to exchange information. This was one of the reasons for the W3C to develop the Extensible Markup Language (XML) [26]. Using this technology, businesses would agree to exchange data on a certain xml-based format or even follow public standards for data interchange such as ebXML [22]. The high acceptance of XML led to the development of software components that exclusively take this language as input, and also produce XML output: XML-based web services. These services are, furthermore, distributed in nature, asynchronous, low-coupled and platform-independent. Their composition enabled the implementation of interoperable business processes. To standardize the specification of these compositions, IBM and other companies proposed a language called BPEL [14] that later became an OASIS standard [23]. All these technological changes have been encouraged by an increasing investment in web services software worldwide, which doubled from 2003 to 2004, reaching $2.3 billion. That figure is expected to continue to grow and become $15 billion by 2009, according to IDC research studies [15].

However, this increasing interest in web services has also led to concerns regarding their trustworthiness. For example, Leavitt [16] highlights the differences between competing standards and Zhang [27] the lack of appropriate development and testing processes. In relation to the latter, the main characteristics of web services influencing testing activities are their asynchronous behaviour, distribution, availability and the lack of user interface.

Bearing in mind these features, the goal of this paper is to describe a model-based testing method to obtain test case specifications for compositions of web services. The BPEL language is introduced in Section 2 along with the composition that will be used as a sample throughout the paper: the “loan approval” composition. An overall description of the test generation method is then given in Section 3. A procedure is described in Section 4 to transform the composition specified in BPEL into a PROMELA model, which is the input language of the SPIN model checker [12]. After that, in Section 5, the approach employed to generate test cases specifications is explained and applied to the sample composition. To end the paper, Section 6 expounds related work and Section 7 details the conclusions and future work.

2 Specification of web services compositions with BPEL

BPEL specifications represent the behaviour of business processes based on compositions of web services. They are XML documents composed of two main sec-

```
<process name="loanapproval" [...]>
  <variables>
    <variable name="riskAssessment"
               messageType="asns:riskAssessmentMessage"/>
  [...]
  </variables>
  <partners>
    <partner name="customer" [...]/>
    <partner name="assessor" [...]/>
    <partner name="approver" [...]/>
  </partners>
  <flow>
    <links>
      <link name="receive-to-assess"/>
      <link name="assess-to-setMessage"/> [...]
    </links>
    <receive name="receive1" partner="customer" [...]>
    [...]
    </receive>
    <invoke name="invokeAssessor" partner="assessor"
          portType="asns:riskAssessmentPT"
          operation="check"
          inputVariable="request"
          outputVariable="riskAssessment">
      <target linkName="receive-to-assess"/>
      <source linkName="assess-to-setMessage"
              transitionCondition=
              "bpws:getVariableData('riskAssessment',
              'risk')='low'"/>
      <source linkName="assess-to-approval"
              transitionCondition="
              bpws:getVariableData('riskAssessment',
              'risk')!='low'"/>
    </invoke> [...]
  </flow>
</process>
```

Fig. 1. Extract from the "loan approval" BPEL specification

tions: declarations and the specification of the business process itself. In the declarations part, the `partners` are identified: each partner stands for a web service that participates in the business process. Other elements included in this first part are the `variables`, which enable the intermediate storage of values.

The specification of the business process consists of a set of activities that can be executed. These activities may be either basic or structured. Among the first, the business process can invoke web services or provide operations by means of the `invoke` and `receive` activities. It can also update the values of the variables using the assign construct. Structured activities prescribe the order in which a collection of activities take place. For example: a `sequence` activity establishes a sequential order and a `while` forces the repetition of the execution of a set of activities until a given statement becomes false.

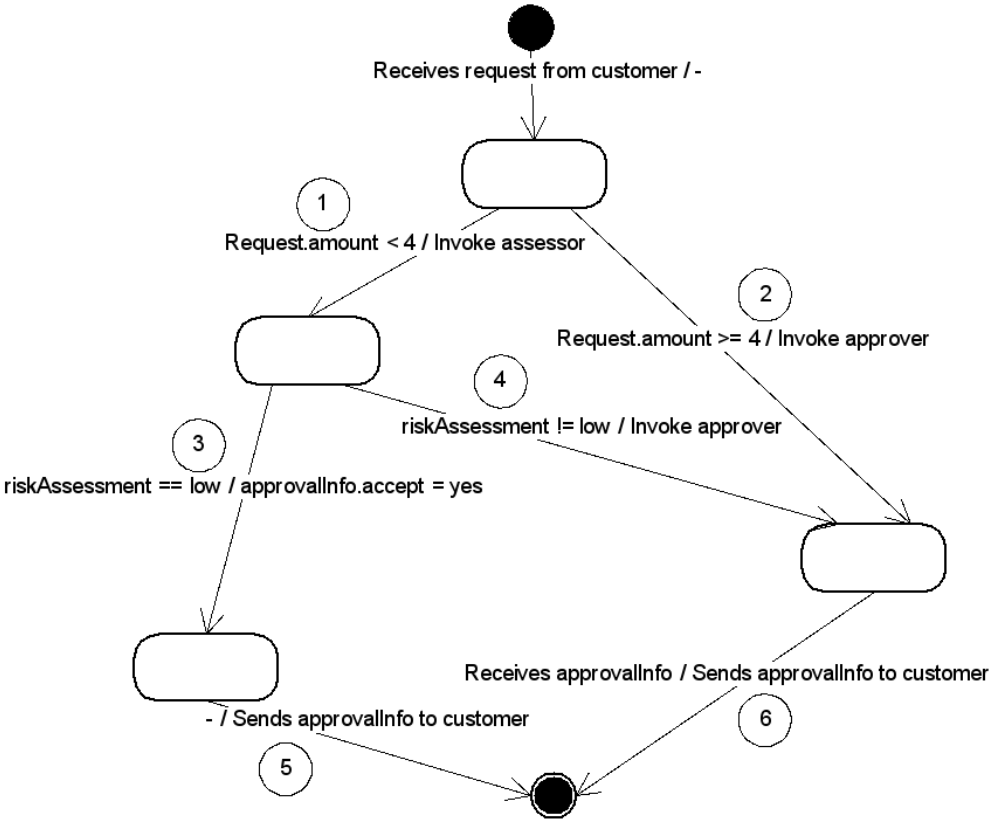

Fig. 2. Representation of the BPEL process described in the "loan approval" sample

A structured activity that is not so common in other languages is the `flow`. Activities grouped in such a construct are concurrent, and so a flow completes when all of its activities have completed. BPEL activities may also be assigned a construct that expresses synchronization dependencies between them: `link`. If activity A is one `source` of a link and activity B is one of its `targets`, that means B must be executed after A. Links may also be given a `transitionCondition` attribute that specifies a necessary condition for the link to become active.

A simplified version of the sample BPEL composition called “loan approval” is outlined in Figure 1. It was published within the specification of the standard [14] and is frequently applied to validate research on web services development. The goal of this business process is to conclude whether a certain request for a loan will be approved or not. To do so, it receives a request from a partner called “`customer`” and invokes two other partners. The “`assessor`” partner measures the risk associated with low amount requests. Another partner, called “`approver`”, approves requests that are either made for a large amount of money or which are evaluated by the assessor as not having a low risk. Figure 2 represents the behaviour of this business process.

3 Overview of the generation of test cases specifications

Model checking [4] is a formal verification technique that enables the automatic detection of whether certain properties hold in a model. It has lots of well documented applications, ranging from the verification of protocols [25] to fault detecting in software systems [11]. SPIN is one of the most commonly used model-checking tools. Using SPIN, properties can be specified by assertions in the model or shaped as Linear Temporal Logic (LTL) formulae. The tool searches all the possible states within the model and checks whether the properties hold. If not, it gives a trace of the steps illustrating the violation of the property, which is called a counterexample. Model checking is commonly used for systems verification, but it can be applied to generate test cases [24, 10]. In order to obtain a test case for a certain condition C, the model checker is fed with a model for the software and a LTL formula stating that C never holds. The output obtained from the tool is hence a counterexample in which the software fulfils C. That counterexample can be transformed into a test case, as it describes an execution of the software in which the desired test condition holds.

The above technique can be applied to generate test case specifications for BPEL compositions. Firstly, the Composition Under Test (CUT) is transformed into PROMELA. Then, in order to produce test cases, test requirements are identified using a transition coverage criterion. To do so, transitions are identified in the BPEL specification, whether explicit or implicit, and are mapped onto the model. In addition, each transition is expressed in terms of a LTL property expressing that "`the transition X is never executed`". The counterexample obtained from a SPIN run is thus a sample execution of the BPEL process in which at least the transition included in the LTL is exercised. To obtain a set of test cases that provides transition coverage, the tool is repeatedly executed with each previously identified transition. This method is depicted in Figure 3 and will be described in further detail in the following sections.

4 Transforming BPEL specifications into PROMELA

As explained in Section 2, BPEL specifications express the behaviour of a business process. This behaviour is modelled, in PROMELA, into a `process` construct. The specifications also include partners, which are the different web services. They are also transformed into PROMELA processes: one is included for each partner in the specification.

The business process and the partners communicate through portTypes, which are transformed into PROMELA message channels as in the following example:

```
chan loanassessor_riskPort_IN = [QLENGTH] of
                                {byte, byte, byte};
```

In the above example, a channel called `loanassessor_riskPort_IN` is declared with a maximum length of QLENGTH messages and supporting messages

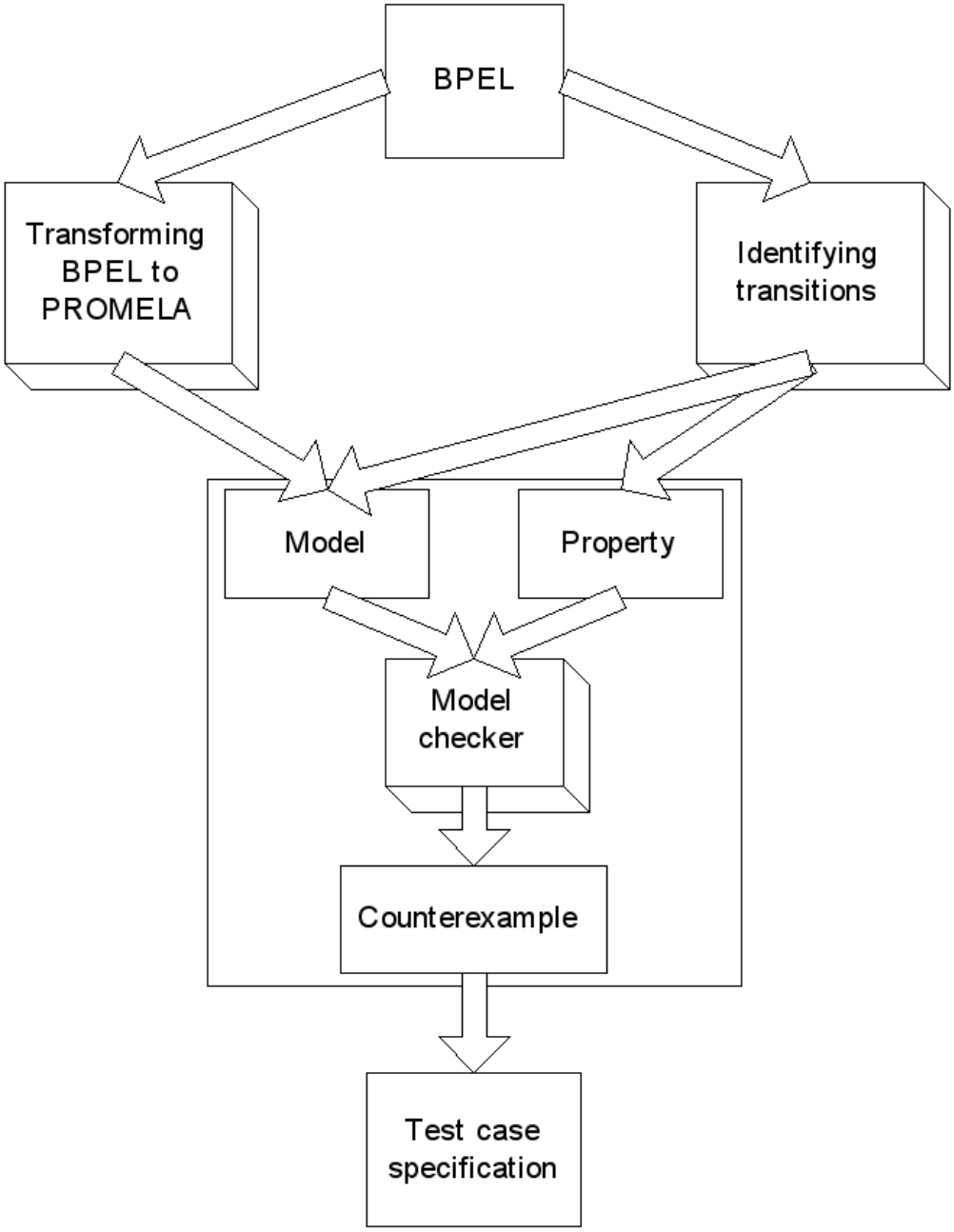

Fig. 3. An overview of the proposed method

consisting of three attributes of type `byte`. Each portType will have two channels: one for input messages and another for output.

Message types are declared as `typedefs` in PROMELA (a construct similar to "record" in the C programming language). BPEL data types are discretized from the original ones: every simple data type becomes byte except for boolean, which stays the same.

Each activity must be appropriately modelled to represent the behaviour of the business process. For example, `invoke` and `receive` activities use the `!` and `?`

channel operators that send and receive messages to and from the channels. Therefore, the statement `BPEL_loanApprovalPort_OUT! approvalInfo.accept` sends a message containing the variable `approvalInfo.accept` to channel `BPEL_loanApprovalPort_OUT`, which is the representation of the output portType of the business process in the "loan approval" sample.

In our approach, the behaviour of the partners is modelled using the BPEL specification as the only input. This is one of the major differences with other lines of work such as [8]. To do so, two sources of information about the partners' behaviour exist in BPEL. Firstly, the kinds of operations that the partners provide or request and the exchanged data are explicit in the specification. Secondly, in a more indirect way, BPEL specifications include how the business process handles the information received from or sent to the partners. This information enables us to build a simple PROMELA model (a mock) for each partner, which is incorporated into its PROMELA process. For example, the business process of the "loan approval" example receives `riskAssessment.risk` from partner `assessor`, and then examines whether it is `low` or not. Consequently, `assessor` is modelled to reply `low` and `other` in order to exercise both conditions. As a result, the behaviour incorporated into the PROMELA process for a partner follows these rules:

(i) If the BPEL specification has no reference to the data, it will be given an `undefined` value;

(ii) if the data is compared to a numerical constant, it will be given the value of the constant, a lower value and a higher third value;

(iii) if the data is discrete, it will be given each of the discrete constants in the BPEL specification and a value different from them, called `other`.

After the BPEL specification is modelled in PROMELA, transitions are identified in the specification and mapped within the model. Two kinds of transitions are distinguished: implicit and explicit. The first ones are obtained from activities that impose at least two possible execution paths. For example, an `invoke` activity may be defined to receive a reply from the invoked web service or not, so two transitions can be identified. Other activities with implicit transitions are `flow` or `while`. Explicit transitions are taken, in the composition, from `link` constructs, as they explicitly establish transitions between activities that have them as `source` or `target`.

5 Using SPIN to generate test case specifications

The test cases generated with SPIN will be systematically selected to satisfy a transition coverage criterion for the BPEL specification. Specifically, the criterion (taken from [20]) states that the resulting test suite must include test cases that cause every transition in the BPEL specification to be taken. To generate test cases specifications for the CUT that satisfy such a criterion, two different steps are required. First of all, LTL properties must be specified for SPIN to find counterexamples in which the PROMELA transitions are covered. Secondly, counterexamples must be

transformed into test cases specifications for the CUT.

All the identified transitions are uniquely differentiated by means of a number and, in PROMELA, a boolean variable called `tranX` (where X is the number of the transition) is defined for each of them. The variable will take a positive value in the case of the transition being exercised by the model checker. Those variables are employed to specify a LTL property and find a counterexample for one given transition. Namely, for transition X, the LTL is "`[] !tranX`", expressing that the variable associated with that transition is always false. After the model checker has been executed, a certain counterexample is given in which the transition is exercised.

In order to complete the test case specification, a difficulty arises when trying to interpret the expected output of the test case. The obtained counterexamples give only the sufficient indications so as to reach a state in which the property is false, but do not continue the execution of the model to its end. Therefore, to get the expected output for the test case, a boolean variable is included within the model, indicating that the business process ends. Thus, the specifications of the properties change to LTL formulae such as:

"`[](!tranX || !bpel_ends)`"

Lastly, to obtain the test case specification, it suffices to take the operations with channels. These operations give a complete description of the test case inputs and the output, as they represent the information exchanged between the business process and the partners.

To build a test suite that meets the above defined transition coverage criterion, the model checker is executed as many times as transitions are identified in the BPEL. To reduce the number of test cases, all the transitions covered with each counterexample are taken into account. In the described model, this is accomplished by inspecting the values of all the variables associated with transitions.

Applying the method to the "loan approval" composition, the numbered transitions in Figure 2 are identified and mapped into PROMELA. Using SPIN with the model obtained from the BPEL and completing the identified transitions, we need three executions to obtain transition coverage. On the first run, the model checker is fed with the LTL "`[](!tran1 || !bpel_ends)`" and so it produces the counterexample represented in Figure 4. In this figure, the partner that executes the step is shown on the left – customer, bpel or assessor. The counterexample covers transitions «1, 3 and 5» and is transformed into a test case specification with two inputs:

(i) the customer makes a request for an amount of 3 (less than four)

(ii) the risk assessment from the assessor is low

and one expected output: the reply to the customer is affirmative.

After this first result, SPIN is run twice again, finding counterexamples that cover transitions «2 and 6» and «1, 4 and 6», respectively. These counterexamples are transformed into test cases as in the example presented above. In this case

```
customer: request.amount = 3
customer: BPEL_loanApprovalPort_IN!request
bpel: request.amount<4
bpel: tran1 = true
bpel: loanassessor_riskAssessmentPort_IN!request
assessor: riskAssessment.risk = low
assessor: loanassessor_riskAssessmentPort_OUT! riskAssessment
bpel: tran3 = true
bpel: approvalInfo.accept = yes
bpel: tran5 = true
bpel: BPEL_loanApprovalPort_OUT!approvalInfo
bpel: bpel_ends = true
```

Fig. 4. Extract from a counterexample obtained by SPIN

study, the number of test cases obtained is the minimum required to give transition coverage for the specification.

As regards the performance of the tool, the execution of the verification ends in less than one second on a Pentium4 (3.0 GHz) system with 2 GB of RAM memory, using 32 internal states with a state-vector of 96 bytes to represent the model.

6 Related work

Research in verification and validation applied to compositions of web services may be basically classified in two categories: papers describing formal verification approaches and others that use testing techniques.

Related work describing verification approaches is quite common. The goal is to decide whether it may be said that certain properties hold in the composition under study. These approaches thus share with ours the use of verification tools and the need to build a model for the composition. Fu et al [8] define such a model and also use SPIN to formally verify compositions of web services specified in BPEL. They subsequently enhanced their research to define formal criteria for the feasibility of automated verification applied to compositions of web services [9, 3]. In the same line of work, Foster et al [6] use Finite State Processes (FSP) to model compositions of web services and describe the use of the LTSA tool [7] to formally verify BPEL specifications. They propose specifying the desired properties in terms of Message Sequence Charts, a technique included in the Unified Modelling Language (UML). Lerner [18] also uses the LTSA tool to analyze business processes specified in Little-JIL. Using a different model and verification paradigm, Narayanan and McIlraith [19] propose annotating web services with semantic descriptions (DAML-S) regarding their capabilities to subsequently encode these in a Petri Net. Another, not so closely related paper by Arias-Fisteus et al [1] describes an intermediate formalism (Common Formal Model or CFM) that can be used to specify business processes or to translate existing specifications into it. Subsequently, CFM models are automatically transformed into the input languages of existing verification tools.

There are not many research works on the definition of testing methods for web

services. Chun and Offutt [17] and Offutt and Xu [21] describe the application of mutation analysis and data perturbation in the testing of web services. Their processes are defined at the unit level, so the targets are the individual web services and not their composition. Bertolino and Polini [2] propose a framework for dynamic testing of web services interoperability. They introduce a test phase (an audition) before the services are published on a UDDI registry. In combination with verification techniques, Huang et al [13] describe a method whose goal is similar to ours: the testing of composite web services. They also apply a model checker, but the main differences lie in the input and the testing criteria: they explicitly specify the web services behaviour (using OWL-S) and define the desired properties by hand.

7 Conclusions and future work

In this paper, a model-based method is expounded for obtaining test cases specifications from BPEL compositions of web services to fulfil a transition coverage criterion. The method relies on a model checking tool (SPIN) to automatically obtain test cases specifications from a model of the BPEL process and partners. LTL properties are properly constructed for the resulting test cases specifications to cover transitions identified in the input. After repeated execution of the tool, a test suite is obtained that achieves transition coverage. In the case of the specification having unreachable transitions, the model checker automatically detects these by not providing a counterexample and performing a full verification. One of the advantages of the proposed method is its independence from the particular implementation, as the only required input is the BPEL specification.

An immediate line of future work on the described method is the application of different test criteria, such as those described by Offutt et al in [20] and its automation. Further research is likewise needed to fully determine the scalability of the method and, as there are no publicly available real-life specifications [5], synthetic ones may need to be constructed and tested. To automatically build test cases that are directly executable, the model could be enhanced so as to include information about the partners' particular implementation.

References

[1] Jesús Arias-Fisteus, Luis Sánchez Fernández, and Carlos Delgado Kloos. Formal verification of bpel4ws business collaborations. In *Proceedings of 5th International Conference on E-Commerce and Web Technologies*, volume LNCS 3182, pages 76–85, Zaragoza, Spain, August 31-September 3 2004. Springer Verlag.

[2] Antonia Bertolino and Andrea Polini. The audition framework for testing web services interoperability. In *Proceedings of 31st EUROMICRO*, pages 134–142, Porto, Portugal, 30 August - 3 September 2005.

[3] Tevfik Bultan, Xiang Fu, and Jianwen Su. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1):18–25, 2006.

[4] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000.

[5] Jianchun Fan and Subbarao Kambhampati. A snapshot of public web services. *SIGMOD Record*, 34(1):24–32, 2005.

[6] Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *Proceedings of 18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pages 152–163, Montreal, Canada, 6-10 October 2003.

[7] Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. Tool support for model-based engineering of web service compositions. In *Proceedings of IEEE International Conference on Web Services (ICWS 2005)*, pages 95–102, Orlando, FL, USA, 11-15 July 2005.

[8] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *Proceedings of the 13th international conference on World Wide Web (WWW 2004)*, pages 621–630, New York, NY, USA, May 17-20 2004.

[9] Xiang Fu, Tevfik Bultan, and Jianwen Su. Synchronizability of conversations among web services. *IEEE Transactions on Software Engineering*, 31(12):1042–1055, 2005.

[10] Elsa L. Gunter and Doron Peled. Model checking, testing and verification working together. *Formal Aspects of Computing*, 17(2):201–221, 2005.

[11] Klaus Havelund, Michael R. Lowry, and John Penix. Formal analysis of a space-craft controller using spin. *IEEE Trans. Software Eng.*, 27(8):749–765, 2001.

[12] Gerald J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[13] Hai Huang, Wei-Tek Tsai, Raymond Paul, and Yinong Chen. Automated model checking and testing for composite web services. In *Proceedings of 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pages 300–307, Seattle, WA, USA, 18-20 May 2005.

[14] IBM. Business process execution language for web services version 1.1. On-line. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[15] IDC. IDC research reports. On-line. http://www.idc.com/

[16] Neal Leavitt. Are web services finally ready to deliver? *IEEE Computer*, 37(11):14–18, 2004.

[17] Suet Chun Lee and Jeff Offutt. Generating test cases for xml-based web component interactions using mutation analysis. In *Proceedings of 12th International Symposium on Software Reliability Engineering (ISSRE 2001)*, pages 200–209, Hong Kong, China, 27-30 November 2001.

[18] Barbara Staudt Lerner. Verifying process models built using parameterized state machines. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 274–284, Boston, Massachusetts,USA, July 11-14 2004.

[19] Srini Narayanan and Sheila A. McIlraith. Analysis and simulation of web services. *Computer Networks*, 42(5):675–693, 2003.

[20] Jeff Offutt, Shaoying Liu, Aynur Abdurazik, and Paul Ammann. Generating test data from state-based specifications. *The Journal of Software Testing, Verification and Reliability*, 13(1):25–53, 2003.

[21] Jeff Offutt and Wuzhi Xu. Generating test cases for web services using data perturbation. *ACM SIGSOFT Software Engineering Notes*, 29(5):1–10, 2004.

[22] Organization for the Advancement of Structured Information Standards (OASIS). *ebXML*.

[23] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WSBPEL).*

[24] P.Ammann, P.E.Black, and W.Majurski. Using model checking to generate tests from specifications. In *Procceedings of 2nd IEEE International Conference on Formal Engineering Methods*, pages 46–56, Brisbane, Australia, 1998.

[25] A.W. Roscoe and Philippa J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(1):147–190, 1999.

[26] W3C. *Extensible Markup Language (XML) 1.0 (Third Edition).*

[27] Jia Zhang. Trustworthy web services: Actions for now. *IEEE IT Pro*, pages 32–36, January/February 2005.